Abstract. In this article we introduce a non-monotonic reasoning engine, i.e., the assumption-based truth maintenance system, and two reasoning paradigms for solving the diagnosis problem that are based on this engine. The objective of the article is to present solutions for problems occurring in classical expert systems based on first-order or propositional logic. In particular, we show how to handle inconsistencies of theories by introducing assumptions or hypothesis. As a consequence, some sort of common sense and default reasoning can be solved. The idea is to allow inference only if the inferred results are not leading to a contradiction.

1 Introduction

For more than 50 years Artificial Intelligence (AI) has been established as an independent research field in Computer Science. Artificial Intelligence as a field is more than 10 years older than Software Engineering and the general objectives are much older. It has always been of very much interest of men to explain cognition and reasoning. But not only explanations for cognitive capabilities have been the driving force. Also the idea of constructing automata that solve problems like mathematical calculations or other practical problems usually done by humans is a major driver of the field. In the 70th and 80th of the last century the field of AI had been very popular leading to many practical contributions like expert systems and machine learning but the very high expectations in the field were not achieved. Today AI is a mature field with many contributions used and adapted in other fields like Software Engineering, Computer Graphics and Visioning, and even Computer Games.

Because AI is a very huge field, we have to focus on one particular area. For this article we discuss the area of logic-based AI and in particular non-monotonic reasoning. The objective behind the area is the use of logic for knowledge representation and reasoning. Especially default and common sense reasoning is of interest. The application areas include diagnosis of hardware systems and even program debugging. Of course it is not possible to present and discuss the whole area. Instead we discuss problems occurring when using logic as bases for knowledge representation and reasoning, and present one solution for some of the problems. For this purpose we introduce the assumption-based truth maintenance system. Based on this system we further discuss 2 solutions to the diagnosis problem, i.e., model-based diagnosis and abduction. The objective here is not to state the whole theory behind those reasoning approaches but to explain them briefly and to give algorithms.

It is worth noting that in practice the main problem when using these methods is not due to understanding the theory or the lack of implementations but the development of usable models, which has turned out to be a very difficult and time consuming task. Hence, using logic for
modeling of systems is something we should focus on. This is especially important when considering logic not only to be useful for AI but also for other fields like Software Engineering where logic plays an important rule when it comes into program verification. Since program verification is necessary for constructing very reliable systems, logic is more and more important even in industry and not only in academia.

When reading this article it is required to know the basic principles of first-order logic (FOL) and propositional logic including inference and deduction. To be somehow self contained we will give a very brief overview in the following subsection. However, further readings are maybe required. We recommend [CL73] for a more detailed introduction.

This paper is organized as follows. We first give a short overview on classical logic. In the next section we discuss problems arising when using classical reasoning and propose the assumption-based truth maintenance system as one solution. This section is followed by a discussion on model-based diagnosis followed by an introduction on abductive reasoning. Finally, we conclude the paper.

1.1 Logic - a very brief introduction

Logic is used to represent textual information in a formal way in order to give the information a precise meaning and to remove ambiguity. For example, we might want to express that every day when it rains, the streets are wet. In this sentence we have a quantification, e.g., ”every day” and something that should happen under a certain condition. In first-order logic (FOL) we might express the fact that it is raining at a specific day using a predicate \( \text{rain} \) with an argument representing the date when it is raining, e.g., the fact that it is raining on August, 24th, 2009 might be expressed with the following predicate \( \text{rain}(24082009) \). We also are able to express the sentence about rain and the conclusion that the streets are wet.

\[
\forall X : \text{rain}(X) \rightarrow \text{streets_wet}(X)
\]

In this logical sentence we have a quantifier stating that something is true always, i.e., \( \forall X \). Then we see that the streets are wet when it is raining. This is represented by an implication (\( \rightarrow \)) that is true either if it is not raining or when it is raining and the streets are wet. Note that the truth status of the sentence has to be proved using observations from the world. Hence, if we find a place where it is raining but where the streets are not wet, then the sentence is obviously false. Otherwise, if the sentence cannot be falsified, we can use it to derive other facts.

For example, if we know that \( \text{rain}(24082009) \) because we observed the rain at the given date, then we infer \( \text{streets_wet}(24082009) \), which is not known before. Hence, logical reasoning can be used to derive new facts.

In the following we define the syntax and semantics of FOL and give a basic reasoning schema, i.e., resolution calculus, for a subset of FOL.

**Definition 1 (Term).** We define terms inductively over the set of variables \( V \), constants \( C \), and function symbols \( F \).

- All variables \( v \in V \) are terms.
We define the syntax of FOL inductively as follows:

**Definition 3 (Syntax of FOL).**

- All constants \( c \in C \) are terms.
- If \( f \in F \) is a function symbol and \( t_1 \) to \( t_k \) are terms, then \( f(t_1, \ldots, t_k) \) is a term.

**Definition 2 (Predicates).**

Let \( p \) be a predicate symbol and \( t_1 \) to \( t_k \) be terms, then \( p(t_1, \ldots, t_k) \) is a predicate.

**Definition 3 (Syntax of FOL).** We define the syntax of FOL inductively as follows:

- \( \text{true} \) and \( \text{false} \)
- \( p \) is a sentence if \( p \) is a predicate.
- \( \neg P \) (Negation) if \( P \) is a sentence
- \( P \land Q \) (And) if \( P \) and \( Q \) are sentences
- \( P \lor Q \) (Or) if \( P \) and \( Q \) are sentences
- \( P \rightarrow Q \) (Implication) if \( P \) and \( Q \) are sentences
- \( P \leftrightarrow Q \) (Bi-Implication) if \( P \) and \( Q \) are sentences
- \( \forall X : P \) (All-quantification) if \( P \) is a sentence
- \( \exists X : P \) (Exists-quantification) if \( P \) is a sentence

We now define the semantics of FOL using an interpretation function \( I \). We first define this function for terms.

**Definition 4 (Semantics of terms).** Let \( I \) be an interpretation. The semantics of terms is defined as follows:

- For variables \( v \in V \): \( I(v) = v \).
- For constants \( c \in C \): \( I(c) = c_0 \) where \( c_0 \) is a constant that corresponds to \( c \).
- For general terms \( f(t_1, \ldots, t_k) \): \( I(f(t_1, \ldots, t_k)) = f_0(I(t_1), \ldots, I(t_k)) \) where \( f_0 \) is the corresponding function symbol of the interpretation.

Using the definition of the interpretation of terms we are now able to define the semantics of FOL sentences.

**Definition 5 (Semantics of FOL).** Let \( I \) be an interpretation function returning \( T \) if the logical sentence is true and \( F \), otherwise.

- \( I(\text{true}) = T \) and \( I(\text{false}) = F \)
- Predicates \( p(t_1, \ldots, t_k) \): \( I(p(t_1, \ldots, t_k)) = T \) if there exist a \( p_0(I(t_1), \ldots, I(t_k)) \) in the interpretation, and \( F \), otherwise.
- \( I(\neg P) = T \) if \( I(P) = F \) and \( F \), otherwise.
- \( I(P \land Q) = T \) if \( I(P) = T \) and \( I(Q) = T \) and \( F \), otherwise.
- \( I(P \lor Q) = T \) if either \( I(P) = T \) or \( I(Q) = T \) and \( F \), otherwise.
- \( I(P \rightarrow Q) = T \) if \( I(P) = F \) or if \( I(P) = I(Q) \) and \( F \), otherwise.
- \( I(P \leftrightarrow Q) = T \) if \( I(P) = I(Q) \) and \( F \), otherwise.
- \( I(\forall X : P(X)) = T \) if for all replacements of variable \( X \) with a variable-free term \( c \) \( I(P(c)) = T \), and \( F \) otherwise.
- \( I(\exists X : P(X)) = T \) if there exists a replacement of variable \( X \) with a variable-free term \( c \) such that \( I(P(c)) = T \), and \( F \), otherwise.
In this definition we separate the syntactical part of FOL from the space of interpretation. Usually, the predicates, constants and functions are interpreted by themselves. Such an interpretation is called Herbrand interpretation.

We now discuss how to derive new knowledge from existing one. For this purpose we introduce the definition of model.

**Definition 6 (Model).** Every interpretation \( I \) that makes a given sentence \( P \in \text{FOL} \) true is called a model of \( P \).

We now define how consequences can be derived from logical sentences.

**Definition 7 (Consequence).** Let \( P \) and \( Q \) be FOL sentences. \( Q \) can be derived from \( P \) if and only if every model of \( P \) is also a model of \( Q \). Formally, we write \( P \models Q \).

From entailment we define logical equivalence.

**Definition 8 (Equivalence).** Two FOL sentences \( P \) and \( Q \) are equivalent if we can entail \( P \) from \( Q \) and vice versa, i.e., \( P \models Q \iff Q \models P \).

Using entailment as defined before we are able to infer new knowledge. However, if we want to automate the entailment, then we have to find simple rules. Such rules form a resolution calculus. In the following we describe the resolution calculus only for a subset of FOL, i.e., the subset of propositional horn clauses. In this subset we do not allow any variables. Moreover, we only allow rules of the form

\[
P_1 \land \ldots \land P_k \rightarrow P_{k+1}
\]

where \( P_i \) are predicates comprising no variables. We also say that such predicates are grounded. Note that facts are represented as rules where \( k = 0 \) such that the left side of the implication is empty. Moreover, we allow sentences comprising single predicates to be negated. For simplicity the all-quantifiers are usually not stated anymore. Moreover, the rules are usually given in form of clauses. A clause comprises only predicates or their negation. The stated rule is represented as clause as follows:

\[
\{\neg P_1, \ldots, \neg P_k, P_{k+1}\}
\]

The following resolution rule makes use of clauses and states that if we have a clause of the from \( A \cup \{P\} \) and another one \( B \cup \{\neg P\} \), then we are able to derive the clause \( A \cup B \). Formally, we express this resolution step as follows:

\[
\begin{align*}
A \cup \{P\} & \quad B \cup \{\neg P\} \\
\hline
A \cup B
\end{align*}
\]

Let us use this resolution rule on our previous example. We know that:

\[
\forall X : \text{rain}(X) \rightarrow \text{streets\_wet}(X) \\
\text{rain}(24082009)
\]
When replacing the variable $X$ with $24082009$ we obtain a grounded, i.e., variable free, sentence:

$$rain(24082009) \rightarrow streets\_wet(24082009)$$

This sentence together with the fact can be used infer the desired information:

$$\{\neg rain(24082009), streets\_wet(24082009)\} \cup \{rain(24082009)\}$$

This example concludes this very short introduction into FOL. However, it is worth noting that the given resolution rule is a general one, which can be easily adapted for general FOL. For this purpose we have to define substitution of variables with terms and finding such a substitution that makes two predicates syntactical equivalent. The latter is called unification.

### 2 Problems with classical reasoning

In this section we discuss some important problems regarding the use of classical reasoning based on logics and in particular FOL. All of the problems support the need for alternative logical reasoning systems, which we discuss later on. The discussed problems can be classified into 2 categories. The first category comprise cases dealing with inconsistencies arising when new information is available. The second one is due to the use of logic for various types of reasoning, e.g., explanatory reasoning.

We start with the discussion on theories leading to inconsistencies. Let us consider the following example.

**Example 1 (Nixon diamond).** Quakers are members of the religious movement of the Religious Society of Friends. Members of the movement generally take actions for peace and against participation in war. Republicans are members or supporters of the United States Republican Party, which are not usually no pacifists. This textual knowledge can be formalized using FOL as follows:

$$\forall X : (quaker(X) \rightarrow pacifist(X)) \quad (1)$$

$$\forall X : (republican(X) \rightarrow \neg pacifist(X)) \quad (2)$$

From Richard Nixon’s biography we know that he was a Quaker and a Republican.

$$quaker(nixon) \land republican(nixon) \quad (3)$$

When taking all three sentences together, we easily see that the resulting theory is inconsistent. This inconsistency comes from the fact that classical FOL allows not to formulate sentences with vague knowledge. In this example, it might not be case that all Quakers are pacifists but
usually they are. The same holds for Republicans. In order to solve the problem a quantification stating that all except some entities have a certain property. Another solution would be to explicitly add the information regarding special entities to the general rules, e.g.:

\[ \forall X : (\text{quaker}(X) \land X \neq \text{nixon} \rightarrow \text{pacifist}(X)) \]  

(4)

We do not need to change the second rule regarding Republicans because of the membership of Nixon in the US Navy and his involvement in the Vietnam war, which indicates that he was not a pacifist.

Note that the name of the problem, i.e., Nixon diamond comes from the graphical representation of the inferences leading to the inconsistency. In particular, we use (1) and (3) to infer \( \text{pacifist}(\text{nixon}) \), and (2) and (3) to infer \( \neg\text{pacifist}(\text{nixon}) \), which finally leads to the inconsistency.

In the last example, we discussed the problem of inconsistency arising when new knowledge is available. For this small example coming up with a simple solution is possible but consider the case of a knowledge base of several thousands of rules where finding those rule leading to a conflict is not that easy anymore. An automated system that is able to handle inconsistencies arising in certain situations is necessary. Moreover, such a system does not only provide a solution to the problem but helps to simplify the knowledge base because default rules, i.e., rules that are true until a conflict shows up, can be stated directly.

The second problem we are discussing is the problem of knowledge re-use. In particular we are not so much interested in directly re-using the rules but in using the same knowledge base for different purposes like prediction, diagnosis, or explanation. We illustrate this problem on the following example.

**Example 2 (Diagnosis of digital circuits).** Let us consider the following simple digital circuits comprising 4 inverters that are connected together to form an inverter chain.

![Diagram of digital circuits](image)

The behavior of the system can be easily expressed using FOL. For this purpose, we first express the behavior of a component, and than formalize the structure of the system. Note that a modeling approach based on component and connections is called component-connection modeling-paradigm. The behavior of an inverter can be stated as follows:

\[ \forall X : \text{inverter}(X) \rightarrow ((\text{in}(X, 0) \leftrightarrow \text{out}(X, 1)) \land (\text{in}(X, 1) \leftrightarrow \text{out}(X, 0))) \]  

(5)

In order to formalize the structure of the circuit we may use the following FOL rules:
\[ \inverter(I_1) \land \inverter(I_2) \land \inverter(I_3) \land \inverter(I_4) \]
\[ \forall X : (\text{val}(a, X) \leftrightarrow \text{in}(I_1, X)) \]
\[ \forall X : (\text{out}(I_1, X) \leftrightarrow \text{in}(I_2, X)) \]
\[ \forall X : (\text{out}(I_2, X) \leftrightarrow \text{in}(I_3, X)) \]
\[ \forall X : (\text{out}(I_3, X) \leftrightarrow \text{in}(I_4, X)) \]
\[ \forall X : (\text{out}(I_4, X) \leftrightarrow \text{val}(e, X)) \]

(6)

In addition we have to add knowledge stating what is not possible, i.e., a connection having the different values at the same time.

\[ \forall X : \neg(\text{val}(X, 0) \land \text{val}(X, 1)) \]
\[ \forall X : \neg(\text{out}(X, 0) \land \text{out}(X, 1)) \]
\[ \forall X : \neg(\text{in}(X, 0) \land \text{in}(X, 1)) \]

(7)

Let \( Th \) comprising the rules from 5, 6, and 7. Using \( Th \) together with the observation \( \text{val}(a, 1) \) we obtained from the real circuit. Using the FOL resolution calculus we are able to derive \( \text{val}(e, 1) \) from \( Th \cup \{ \text{val}(a, 1) \} \). The necessary resolution steps are:

\[
\begin{array}{c}
\text{val}(a, 1) \\
\forall X : (\text{val}(a, X) \leftrightarrow \text{in}(I_1, X)) \\
\inverter(I_1) \\
\forall X : \text{inverter}(X) \rightarrow ((\text{in}(X, 0) \leftrightarrow \text{out}(X, 1)) \land (\text{in}(X, 1) \leftrightarrow \text{out}(X, 0))) \\
\text{in}(I_1, 1) \\
\text{out}(I_1, 0) \\
\text{out}(I_4, 1) \\
\forall X : (\text{out}(I_4, X) \leftrightarrow \text{val}(e, X)) \\
\text{val}(e, 1)
\end{array}
\]

We have shown that a logical theory \( Th \) can be used for predicting output values. Note that the same model can also be used for predicting other values of connections if enough observations are given. However, the model cannot be used for diagnosis purposes. For example assume that we have not only the observation \( \text{val}(a, 1) \) but measure \( e \) to be false, i.e., \( \text{val}(e, 0) \). Obviously, from \( Th \) and \( \{ \text{val}(a, 1), \text{val}(e, 0) \} \) we infer a contradiction! But there is no way of getting valuable information regarding the source of the contradiction. \( \square \)

In this article we will tackle the problem of diagnosis. For this purpose we show that a small modification of the model from the last example, will be sufficient for prediction and diagnosis at the same time using non-monotonic reasoning.

Note that there are more problems concerned with classical logics, for example the ramification problem. This problem is due indirect effects of actions. One example, is the dust lying on the surface of a robot. If the robot is moving the dust might move as well but this is not usually modeled in the right way. Another example would be asking where the leg of the former president of the USA Abraham Lincoln buried. Usually, someone would conclude that the leg is buried at the same place than the rest of Abraham Lincoln’s body. Only we have further knowledge like the loss of a leg because of some reason, we would conclude in a different way. Similar problems
occur when dealing with common sense reasoning. If usually something is true, it should be concluded unless we know better. For example, we usually assume birds to fly but there are birds like penguins that cannot fly. Using this example, we can easily explain the term non-monotonic reasoning.

**Example 3.** We first state that all birds can fly and that one bird is named Tweety.

\[ \forall X : bird(X) \rightarrow fly(X) \]

\[ bird(tweety) \]

From this two pieces of knowledge we can derive that Tweety can fly, i.e.,

\[ \begin{align*}
  & bird(tweety) \\
  & \forall X : bird(X) \rightarrow fly(X) \\
  \hline
  & fly(tweety)
\end{align*} \]

If we add another fact like Tom is also a bird \( bird(Tom) \), then we can also derive that Tom is also able to fly. Hence, more knowledge allows us to derive more new rules and facts. As a consequence we conclude that classical reasoning (in FOL) is monotonic.

The situation changes if we add new knowledge like penguins cannot fly and that Tweety is a penguin.

\[ \forall X : penguin(X) \rightarrow \neg fly(X) \]

\[ penguin(tweety) \]

In this case we derive a contradiction from which we can derive everything. There are no more restrictions on the number of facts to be derived. Therefore, deriving conflicts is not the thing we want to have for knowledge bases when dealing with FOL. An alternative reasoning schema would be to say that we are able to derive something only in case no contradiction can be derived. Such reasoning schema would be non-monotonic because adding new facts and rules might reduce the number of facts to be derivable. In this example, we cannot derive that Tweety can fly when adding the information that Tweety is a penguin. Note that alternatively we cannot derive that Tweety cannot fly.

In the rest of this article we focus on the problem of handling inconsistencies and show how the proposed solution can be used for prediction, diagnosis, and explanation. In particular, we first start with an introduction to the ATMS a reasoning system that is able to deal with inconsistencies. Secondly, we show that the ATMS can be used for diagnosis, and finally we present a solution of abductive reasoning, which implements explanation and diagnosis capabilities. The abductive reasoning algorithm also makes use of the ATMS. It is worth noting that the proposed diagnosis method and abductive reasoning are similar with one exception. The former one only requires models to handle the fault-free behavior, i.e., the ordinary expected behavior of a system, whereas the latter one requires the existence of fault models. Fault models are for explicitly representing what happens in case of a fault. Moreover, abductive reasoning can be used in all cases where cause-effect knowledge is available.
3 The ATMS

The problem of retaining consistency of logical theories can be solved in different ways. First, it is possible to change the formulae in order to remove inconsistencies. The drawback of this method is that it requires changes that are not that easy to perform especially in the case of large theories. Another alternative way is to use an automated system that retains consistency without any manual changes. The idea is to introduce additional propositions that might be true or false depending on consistency considerations. Such propositions are called assumptions. The automated systems sets the truth state of the assumption in order to guarantee consistency. One particular representative of such systems is the Assumption-based Truth Maintenance System (ATMS) [dK86a,dK86b].

![Diagram of the problem solving architecture for the ATMS](image)

Fig. 1. The problem solving architecture for the ATMS

The basic ATMS works with propositional horn clauses, i.e., facts and rules. It is intended to be coupled with a problem solver that implements the domain knowledge and corresponding inference procedures in order to send the results of the inferences to the ATMS. The ATMS takes the facts and rules (called justifications) and adapts the truth state of the underlying assumptions in order to retain consistency. Moreover, the ATMS determines the belief and disbelief of propositions with respect to the given set of assumptions. The architecture of a reasoning system based on the ATMS and a problem solver is depicted in Figure 1. We now show how the combination of the ATMS and the problem solver works using an extended version of the Nixon diamond (Example 1).

**Example 4 (Nixon diamond using assumptions).** The original Nixon diamond (Example 1) can be easily adapted in order to solve the underlying problem. What we have to do is to change the first two rules to:

\[
\forall X : (\text{quaker}(X) \land \text{Default}\_\text{quaker}(X) \rightarrow \text{pacifist}(X))
\]

\[
\forall X : (\text{republican}(X) \land \text{Default}\_\text{republican}(X) \rightarrow \neg \text{pacifist}(X))
\]

In these equation we make explicit whether a quaker respectively a republican behaves in the default and expected way. For this purpose we use the predicates $\text{Default}\_\text{quaker}(X)$ and $\text{Default}\_\text{republican}(X)$.
\textit{Default} \textit{\textbackslash republican}(X). Note that Rules 8 and 9 cannot be send to the ATMS directly because they comprise a variable and the ATMS is only able to handle propositional rules and facts. Therefore, we need a logical reasoning system as a problem solver that takes rules and facts and sends the corresponding justifications to the ATMS. For example, such a theorem prover would be able to derive two propositional rules from 8 and 9:

\[
\text{quaker(nixon)} \land \text{Default\textbackslash quaker(nixon)} \Rightarrow \text{pacifist(nixon)}
\]

\[
\text{republican(nixon)} \land \text{Default\textbackslash republican(nixon)} \Rightarrow \text{not}\textunderscore\text{pacifist(nixon)}
\]

For this rules the variable \(X\) has to be replaced with \(nixon\). Moreover, the negation of \textit{pacifist} is mapped to a new proposition \textit{not\_pacifist}. Note that we use \(\Rightarrow\) instead of \(\rightarrow\) to indicate a justification send to the ATMS. What is missing is stating that someone cannot be pacifist and no pacifist at the same time. For this purpose the problem solver sends the following justification to the ATMS:

\[
\text{pacifist(nixon)} \land \text{not}\textunderscore\text{pacifist(nixon)} \Rightarrow \text{false}
\]

In addition the facts \textit{quaker(nixon)} and \textit{republican(nixon)} are also send to the ATMS. The ATMS takes the given information and computes sets of assumptions that lead to inconsistencies. In this example \textit{Default\textbackslash quaker(nixon)} and \textit{Default\textbackslash republican(nixon)} cannot be true at the same time. This result can be used to state that either Nixon behaves like an ordinary republican or and ordinary quaker. \(\Box\)

Before introducing an algorithm that implements an ATMS we discuss the basic definitions.

\textbf{Definition 9 (ATMS).} An ATMS is a tuple \((N, A, J)\) where

- \(N\) is a set of nodes including the special node \textit{false} (or \(\bot\)) representing an inconsistency. \textit{false} is also called \textit{nogood}.
- \(A \subseteq N\) is a set of assumptions.
- \(J\) is the set of justifications of the form \(n_1 \land \ldots \land n_k \Rightarrow n_{k+1}\) or \(\Rightarrow n_{k+1}\) where \(n_1, \ldots, n_{k+1} \in N\). The left side of a justification is called the antecedence and the right side the consequence.

The definition of the ATMS uses a different notation than used for other resolution calculi. However, nodes can be seen as proposition and justification as propositional horn clauses. The purpose of the ATMS is to retain consistency. This is done in principle by assigning truth values for assumptions. The ATMS does not implement the assignment directly. Instead the ATMS stores sets of assumptions for each node that allow for deriving the node. We call a set of assumptions an \textit{environment} and the set of environments for a node a \textit{label}. We further make use of the following definitions:

\textbf{Definition 10.} Given an ATMS \((N, A, J)\). A node \(n \in N\) holds in an environment \(E \subseteq A\) iff \(n\) can be derived from \(E\) and the current theory \(J\), i.e., \(E \cup J \models n\).

\textbf{Definition 11.} Given an ATMS \((N, A, J)\). An environment \(E \subseteq A\) is inconsistent iff the \textit{false} node (\(\bot\)) can be derived, i.e., \(E \cup J \models \bot\). Furthermore, an environment \(E \subseteq A\) is consistent if it is not inconsistent.
We use these two definitions in order to define the semantics of an ATMS in terms of node labels.

**Definition 12 (Valid label function).** Given an ATMS \((N, A, J)\) and a function \(\Lambda\) mapping nodes to their labels, i.e., \(\Lambda : N \rightarrow 2^A\). \(\Lambda\) is valid iff for all nodes \(n \in N\) the following holds:

- **\(\Lambda(n)\) is consistent** All of \(\Lambda\)'s environments are consistent.
- **\(\Lambda(n)\) is sound** I.e., \(n\) is derivable from every environment \(E \in \Lambda(n)\).
- **\(\Lambda(n)\) is complete** I.e., every consistent environment \(E \notin \Lambda(n)\) for which \(E \cup J \models n\) is a superset of some \(E' \in \Lambda(n)\), i.e., \(E' \subset E\).
- **\(\Lambda(n)\) is minimal** I.e., for every element \(E \in \Lambda(n)\) there exists no subset \(E' \subset E\) from which \(n\) can be derived \(E' \cup J \models n\).

The task of an ATMS is to maintain the node labels in order to ensure that all of them are valid, i.e., an implementation must ensure to compute minimal, consistent, sound, and complete labels for each node.

**Example 5.** We now illustrate the definitions of ATMS using the rules given in Example 4. The ATMS comprises the nodes \(\text{false, quaker(nixon), republican(nixon), pacifist(nixon), not\text{-}pacifist(nixon), Default\text{-}quaker(nixon), and Default\text{-}republican(nixon)}\) where the latter 2 are also assumptions. Furthermore, we have the following justifications:

\[
\begin{align*}
\text{quaker(nixon)} \land \text{Default\text{-}quaker(nixon)} & \Rightarrow \text{pacifist(nixon)} \\
\text{republican(nixon)} \land \text{Default\text{-}republican(nixon)} & \Rightarrow \text{not\text{-}pacifist(nixon)} \\
& \Rightarrow \text{quaker(nixon)} \\
& \Rightarrow \text{republican(nixon)} \\
\text{pacifist(nixon)} \land \text{not\text{-}pacifist(nixon)} & \Rightarrow \text{false}
\end{align*}
\]

A valid assignment of labels to nodes is depicted in the following table:

<table>
<thead>
<tr>
<th>Node (n)</th>
<th>Label (\Lambda(n))</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{quaker(nixon)}</td>
<td>{}</td>
</tr>
<tr>
<td>\text{republican(nixon)}</td>
<td>{}</td>
</tr>
<tr>
<td>\text{Default\text{-}quaker(nixon)}</td>
<td>{\text{Default\text{-}quaker(nixon)}}</td>
</tr>
<tr>
<td>\text{Default\text{-}republican(nixon)}</td>
<td>{\text{Default\text{-}republican(nixon)}}</td>
</tr>
<tr>
<td>\text{pacifist(nixon)}</td>
<td>{\text{Default\text{-}quaker(nixon)}}</td>
</tr>
<tr>
<td>\text{not\text{-}pacifist(nixon)}</td>
<td>{\text{Default\text{-}republican(nixon)}}</td>
</tr>
<tr>
<td>\text{false}</td>
<td>{\text{Default\text{-}quaker(nixon), Default\text{-}republican(nixon)}}</td>
</tr>
</tbody>
</table>

What we see is that Nixon cannot be an ordinary Quaker and Republican at the same time because of the label of the \(\text{false}\) node. Moreover, it is worth noting that each assumption has an environment with itself as the only element. This is due to the idea that assumptions should be able to derive other nodes. Hence, an assumption is justified by itself.
The use of ATMS solves the Nixon diamond problem. Before discussing an algorithm that computes valid node labels, we have a look at the underlying computational complexity of ATMS (from [Rut91]).

**Theorem 1 (Complexity of ATMS).** Computing valid node labels for ATMS is an NP-hard problem.

*Proof.* The proof is done by (1) showing that the ATMS is in NP, and (2) find a polynomial reduction from a known NP-hard problem.

**ad (1):** ATMS must be in NP. Given a particular input, we can guess a set $S$ of propositions of size $k - 1$, set them to TRUE and run the Horn clause deduction in linear time to confirm that no contradiction arises.

**ad (2):** Reduction from the Max Clique Problem (MCP): Given an instance graph $G$, and an integer $k$, we want to find out if $G$ contains as a subgraph a clique of size $k - 1$ or more. Polynomial reduction from MCP to ATMS: $n$ be the number of nodes in $G$. For every $v \in G$ let $y_v$ be a proposition saying $v$ is in the clique. The $y_v$’s are in the set of assumptions $A$ and propositions $X$. Formula $F$ is a conjunction of clauses: For every pairs $(v, w)$ of nodes in $G$ which are not adjacent, add the rule $y_v \land y_w \Rightarrow false$. This means $v$ and $w$ does not belong to the same clique.

**Claim** $G$ contains a clique of size $k - 1$ or more iff there exists a set $S$ of assumptions of size $k - 1$, that, if all set to TRUE will leave $F$ satisfiable.

$(\Rightarrow) G$ contains a clique $V$ of size $k - 1$. Let all $y_v \in S$ where $v \in V$ be TRUE and the rest to FALSE. It is trivial to see that no rule in $F$ fires. Thus, $F$ is satisfiable.

$(\Leftarrow) S$ is a set of $k - 1$ assumptions that, if all set to TRUE, will leave $F$ satisfiable. Let $V_S$ be the set of corresponding nodes $v$, for which $y_v \in S$. We claim that $V_S$ is a clique. Suppose the converse. Then there must be nodes $v$ and $w$ in $V_S$ that are not adjacent in $G$. But then $y_v \land y_w \rightarrow false$ must be in $F$. Hence, $F$ cannot be satisfiable, contradicting our initial assumptions.

This concludes our proof and it follows that computing all valid node labels of an ATMS is NP-hard. □

### 3.1 The ATMS algorithm

In the following we present a basic algorithm implementing the ATMS. Johan de Kleer [dK88] invented the algorithm. The algorithm ensures that the computed node labels are consistent, minimal, complete, and sound. The algorithm is not very effective and there are various improvements. However, the basic ideas behind the algorithms are worth being discussed. The underlying idea is to represent the ATMS as a and-or-graph where each justification represents and and-connection. Justifications that have the same consequent are or-connected in the graph.

For each node $n$ we store the name of the node, the label $A(n)$, and a set of set of nodes where each set of nodes represent the antecedence of a justification where $n$ is the consequent.

The information stored in each node can be logically interpreted as follows. Let $n$ be the node represented by the tuple:
From this tuple we are able to extract the following logical rules:

\[(A_1 \land \ldots \land A_n) \lor (B_1 \land \ldots \land B_m) \lor \ldots \rightarrow n \]
\[(x_1 \land \ldots \land x_k) \lor (y_1 \land \ldots \land y_j) \lor \ldots \rightarrow n \]

The proposed ATMS algorithm uses the graph data structure and modifies the nodes’ labels whenever a new justification is send from the Problem Solver. The ATMS algorithm is an incremental algorithm. At the beginning there is no node available. If one justification is send to the ATMS the algorithm creates nodes in cases the nodes have not been generated before. Afterwards, the new node label for the consequent node are computed using the node labels of the antecedence nodes. In order to ensure the label properties, i.e., consistency, minimality, soundness, and completeness, the change of labels might have an impact on other nodes that are connected to a consequent via a justification. In case the consequent is member of the antecedence of a justification. Hence, the labels have to be propagated through the graph until no new label change occur. Moreover, the labels of assumptions have to set automatically. As already said, assumptions are justified by themselves. Hence, the label of an assumption comprises one set having the assumption itself as the only element. To distinguish assumptions from ordinary nodes (or propositions) we assume the name of assumptions to start with a capitalized character. Ordinary nodes start with a lowercase character.

The following ATMS algorithm is called using \texttt{PROPAGATE}(J, \Phi, \{\}) where \(J\) denotes the new justification, \(\Phi\) indicates the absence of an optional antecedence node, and the set comprising the empty set the new label of the consequent to be computed.

\begin{algorithm}
\caption{PROPAGATE \((x_1, \ldots, x_n \rightarrow x_{n+1}), a, I)\)}
\begin{enumerate}
\item \textbf{[Compute the incremental update]}
\[L = \text{WEAVE}(a, I, \{x_1, \ldots, x_n\}).\] If \(L\) is empty, return.
\item \textbf{[Update label and recur]} \texttt{UPDATE}(L, x_{n+1}).
\end{enumerate}
\end{algorithm}

In the first step of \texttt{PROPAGATE} the new label is computed using the \texttt{WEAVE} function. The second step, which is only executed if there is a new label for the consequent of the new justification, updates the label using the function \texttt{UPDATE}.

The \texttt{WEAVE} function has 3 parameters. The first is an antecedence node (or \(\Phi\) in the case such a node is not available), the second is the already computed label, and the third is a set comprising nodes, which have not been used for computing the label so far.

\begin{algorithm}
\caption{WEAVE\((a, I, \{\})\)}
\begin{enumerate}
\item \textbf{[Termination condition]} If \(X\) is empty, return \(I\).
\item \textbf{[Iterate over the antecedent nodes]} Let \(h\) be the first node of the list \(X\) and \(R\) the rest.
\item \textbf{[Avoid computing the full label]} If \(h = a\), return \texttt{WEAVE}(\(\Phi, I, R\)).
\end{enumerate}
\end{algorithm}
4. **[Incrementally construct the label]** Let $I'$ be the set of all environments formed by computing the union of an environment of $I$ and an environment of $h$'s label.

5. **[Ensure that $I'$ is minimal and contains no known inconsistency]** Remove from $I'$ all duplicates, nogoods, as well as any environment subsumed by any other.

6. Return $\text{WEAVE}(a, I', R)$.

The first step of $\text{WEAVE}$ checks whether the computation is finished. This is the case when there are no nodes (of the antecedence of a justification) are left. The second step is for iterating over the remaining elements of $X$. This is done in an recursive way by calling $\text{WEAVE}$ again at the end of the function. Before calling the $\text{WEAVE}$ a new label is computed by computing the union of the available environments. Afterward duplicates and supersets are removed. The same holds for nogoods detected within the computation. The step for avoiding computing the label is only called when there is a node $a$ given that is not equal to $\Phi$. Note that this check is used via computing $\text{UPDATE}$ to avoid using a label again during computation.

The $\text{UPDATE}$ function has 2 parameter. The first one $L$ is a label, and $n$ is a node.

**Algorithm $\text{UPDATE}(L, n)$**

1. **[Detect nogood]** If $n = \bot$ then call $\text{NOGOOD}(E)$ on each $E \in L$ and return $\emptyset$.

2. **[Update $n$’s label ensuring minimality]**
   (a) Delete every environment from $L$ which is a superset of some label environment of $n$.
   (b) Delete every environment from the label of $n$ which is a superset of some element of $L$.
   (c) Add every remaining environment of $L$ to the label of $n$.

3. **[Propagate the incremental change to $n$’s label to its consequences]** For every justification $J$ in which $n$ is mentioned as an antecedent call $\text{PROPAGATE}(J, n, L)$.

During the computation of $\text{UPDATE}$ first a check for a nogood is done. This is the case when $n = \bot$ or false. In this case the $\text{NOGOOD}$ function is called. Afterwards a valid label is computed for $n$ applying the appropriate methods. Finally, the new label has to be propagated through the graph data structure again. This is done using $\text{PROPAGATE}$ again. This time $n$ is the parameter instead of $\Phi$.

The $\text{NOGOOD}$ function adds the given environment to the label of the nogood node and removes this environment and all supersets from all labels of all nodes. This step ensures consistency of the labels of the other nodes.

**Algorithm $\text{NOGOOD}(E)$**

1. Mark $E$ as nogood (and add it to the label of $\bot$).
2. Remove $E$ and any superset from every node label.

We now continue the Nixon diamond example to show how the incremental ATMS algorithm works in detail.
Example 6. In this example we use the justifications for the Nixon diamond given in Example 5:

1. \(\text{quaker}(nixon) \land \text{Default}_{\text{quaker}}(nixon) \Rightarrow \text{pacifist}(nixon)\)
2. \(\text{republican}(nixon) \land \text{Default}_{\text{republican}}(nixon) \Rightarrow \neg \text{pacifist}(nixon)\)
3. \(\Rightarrow \text{quaker}(nixon)\)
4. \(\Rightarrow \text{republican}(nixon)\)
5. \(\text{pacifist}(nixon) \land \neg \text{pacifist}(nixon) \Rightarrow \text{false}\)

For the construction of the ATMS we assume the given order of the justifications. Figure 2 shows the changes of the graph data structure when calling the \textsc{Propagate} algorithm. The first picture shows the graph after adding Justification 1. Only three nodes and one justification are present. The label of the node \(\text{pacifist}(nixon)\) is empty because there is no knowledge available regarding the truth state of \(\text{quaker}(nixon)\). After adding Justification 3 in Picture 3, the truth status is known and \(\text{pacifist}(nixon)\) has a new label stating that Nixon is a pacifist if behaves like an ordinary Quaker. Finally, in Picture 5 all node labels are set. □

In the following sections we discuss two possible uses of the ATMS in practice. The first one is diagnosis based on models of the correct behavior of a system. For this purpose we have to define diagnosis and relate them with the labels of the nogood. The second one is abductive reasoning that can be used to explain certain observations for a given logical theory.

Examples for Section 3

ATMS 1 Consider a coffee machine with one tank for water and one for coffee beans.

In case of a request the coffee machine should produce coffee if there is enough water and there are enough beans available. Otherwise, no coffee is produced. A simple model of the coffee machine is the following:
Fig. 2. The ATMS updates when adding the justifications for the Nixon diamond example
Request ⇒ request
Water ⇒ water
Beans ⇒ beans
request \land water \land beans ⇒ coffee
coffee ⇒ request
coffee ⇒ water
coffee ⇒ beans
no_coffee \land request \land water ⇒ no.beans
no_coffee \land request \land beans ⇒ no_water
no_coffee \land water \land beans ⇒ no_request
beans \land no_beans ⇒ ⊥
request \land no.request ⇒ ⊥
water \land no_water ⇒ ⊥
coffee \land no.coffee ⇒ ⊥

1. Compute the labels of the nodes using the incremental ATMS algorithm. Assume that the rules are send to the ATMS in the same order than given in the list of justifications.
2. Compute the label update if we know that there is no coffee. In this case the justification ⇒ no_coffee is send to the ATMS.
3. Now compute the label update again when assuming that there is no water in the tank. How does the justification look like?

4 Model-based diagnosis

The idea behind model-based diagnosis is to use a model of a system directly for computing diagnoses. A diagnosis itself is a set of system components that explain an observed misbehavior. I.e., in order to generate diagnoses we have to have a discrepancy between the real behavior of the (physical) system and the behavior that obtained from the model. Model-based diagnosis [Dav84,Rei87,dKW87] relies on a model of the system that is comprised from components and connections. For each component the correct behavior has to be known. Beside hardware diagnosis model-based diagnosis has been also used effectively for software debugging, e.g., [FSW99,MSW00,MSWW02].

In model-based diagnosis (MBD) the diagnosis problem \((SD,COMP,OBS)\) comprises a system model \(SD\), a set of components \(COMP\), and a set of observations \(OBS\). A diagnosis is as already mentioned nothing else than a subset of the set of components with the capability of explaining a misbehavior. In the following definition of diagnosis we do not rely on this explanatory capabilities but instead define a diagnosis as a subset of the set of components that when assumed to behave wrong retracts an inconsistency.

**Definition 13 (Diagnosis).** Given a diagnosis problem \((SD,COMP,OBS)\). A set \(\Delta \subseteq COMP\) is a diagnosis if

\[
SD \cup OBS \cup \{AB(C)|C \in \Delta\} \cup \{\neg AB(C)|C \in COMP \setminus \Delta\}
\]

is consistent. A diagnosis is said to be minimal if no subset is itself a diagnosis.
In this definition we make use of a predicate $AB$ and its negation. This predicate is for explicitly stating that a given component behaves abnormal. In case of the negated $AB$ predicate the component behaves in a correct way. Hence, the model of the system $SD$ has to have rules using the $\neg AB$ to indicate the normal behavior of components. We illustrate such a model extending Example 2.

**Example 7.** Let us consider again the inverter chain given in Example 2. The original rule for stating the behavior is:

$$\forall X : \text{inverter}(X) \rightarrow ((\text{in}(X, 0) \leftrightarrow \text{out}(X, 1)) \land (\text{in}(X, 1) \leftrightarrow \text{out}(X, 0)))$$

We now introduce the negated $AB$ predicate to state the normal behavior and extend the previous rule leading to the following new rule of the knowledge base:

$$\forall X : \text{inverter}(X) \rightarrow (\neg AB(X) \rightarrow ((\text{in}(X, 0) \leftrightarrow \text{out}(X, 1)) \land (\text{in}(X, 1) \leftrightarrow \text{out}(X, 0))))$$

(10)

The other rules of Example 2 remain the same. This is also a reason for using model-based diagnosis. Changes of the component model are local and have no direct influence to other rules or fact. Hence, maintaining the knowledge base is easier and requires less effort.

The set of components for the example is $\{I_1, I_2, I_3, I_4, I_5\}$. One diagnosis for the given observations $OBS = \{\text{val}(a, 1), \text{val}(e, 0)\}$ is $\Delta = \{I_1\}$.

Accordinly to the definition of diagnosis we obtain all minimal diagnoses by computing every subset of $COMP$ and checking consistency. This requires $2^{|COMP|}$ checks and is not very efficient. Moreover, the ATMS cannot be directly used for diagnosis in this case. In order to make use of the ATMS we have to find a different way for computing diagnoses. We start with the definition of conflicts.

**Definition 14 (Conflict).** Given a diagnosis problem $(SD, COMP, OBS)$. A set $CO \subseteq COMP$ is a conflict if and only if $SD \cup OBS \cup \{\neg AB(C) | C \in CO\}$ is inconsistent. A conflict is said to be minimal if none of its subsets is a conflict.

A conflict is a subset of the components that leads to an inconsistency with the given observations when assuming all components in the conflict to be working correctly. Hence, a conflict states which part of the system cannot work correctly at the same time. For the last example $\{I_1, I_2, I_3, I_4, I_5\}$ is the only conflict. For other examples there might be more. The idea of getting from conflicts to diagnoses is the following. Since all conflicts state the some components cannot work correctly and the fact that we have to remove the inconsistency from all conflicts, we have to select a component from every conflict and state the component to behave faulty. Thus we remove all conflicts and the selection is a diagnosis. Informally speaking each such selection is a hitting set. In the next section we express the relationship between diagnoses and conflicts and define hitting sets formally.
### 4.1 Hitting-sets

Before stating the relationship between diagnoses and conflicts we define hitting sets.

**Definition 15 (Hitting-set).** A set \( h \subseteq \bigcup_{x \in F} x \) for a set of sets \( F \) is a hitting-set if and only if the intersection of \( h \) with all elements of \( F \) is not empty, i.e., \( hs(h) \leftrightarrow \forall x \in F : h \cap x \neq \emptyset \).

In this definition a set comprising all elements of every set in \( F \) is a hitting set. But in practice we are not really interested in such large sets. Therefore, we further define minimal hitting sets.

**Definition 16 (Minimal hitting-set).** A hitting-set \( h \) is minimal if and only if no proper subset of \( h \) is itself a hitting set, i.e., \( \minHS(h) \leftrightarrow \nexists h' \subset h : hs(h') \).

Note that the definition of minimal hitting-set is not based on cardinality. For example the set \( \{12\} \) is a minimal hitting-set for the set \( \{\{12,13,15\}, \{12,14,16\}, \{12,14\}\} \) but also \( \{13,14\} \). From here on and if not otherwise stated we are always referring to minimal hitting-sets.

In MBD the relationship between diagnoses and conflicts is summarized in the following theorem. We refer the interested reader to [Rei87] for a proof and more properties following from the underlying definitions.

**Theorem 2.** Given a diagnosis problem \((SD, COMP, OBS)\). Every minimal hitting set of a the set of minimal conflicts for \((SD, COMP, OBS)\) is a minimal diagnosis for the same diagnosis problem.

Hence, whenever knowing all conflicts we are able to compute all diagnoses using hitting sets.

Let us now come back to the construction of an algorithm for computing hitting-sets from arbitrary sets of sets. In 1987 Reiter [Rei87] published such an algorithm, which was improved by Greiner et al. [GSW89] two years latter. The proposed algorithm constructs a directed acyclic graph (DAG) in a breadth first manner from the given set of sets \( F \). The underlying idea is as follows. We first construct a root node and label it with a previously not considered set from \( F \). For each element of this selected set we insert a new node and a new arc connecting this node with the root node. The arc is labeled with the element. For the new node we now have to find a new set from \( F \), that has an empty intersection between the set of arc labels from this node to the root node. If there is no such set, then the node represents the end of the construction and the arc labels to the root node form a new (maybe non-minimal) hitting-set. The construction stops after no new node can be generated.

Of course this description does not handle all cases. For example, it can be the case that there is no need for a new node because the same situation w.r.t. the arc labels has been considered already. In this case we are able to re-use a node. Another case is that we might already have a minimal hitting-set but there might be another branch in the resulting DAG representing a superset of this hitting-set. In this case, we are able to close the node and there is no need for further expanding the node. The third case is due to the fact that \( F \) might has two elements where one is the superset of the other. If we first select the larger set during computation, then we have to prune the DAG later on when using the subset for computation. Note that without pruning
the DAG size is larger than necessary and performance decreases. The third rule is of especial
importance if the set $F$ is not known in advance but extended during the hitting-set computation.
For example, in model-based diagnosis according to [Rei87] this is the case.

When combining the ATMS and the hitting sets for computing diagnosis we know all con-
licts in advance. Therefore, it is always possible to sort $F$ according to the cardinality of the
elements and start constructing the DAG from the smallest element of $F$ to the largest one. The
following algorithm implements exactly this idea. It can be considered as variant of the original
algorithm where we eliminated the pruning rule. This elimination is possible when assuming that
$F$ is a sorted collection with respect to the cardinality of the sets where the left-most element is
the smallest one.

\textbf{ALGORITHM AllMinHittingSets}($F$, $MAX$)
\textbf{Input}: A sorted collection $F$ of sets with respect to cardinality. The smallest set is assumed to
be the left-most element of $F$. A number $MAX > 0$ which specifies the maximum size of the
generated hitting sets.
\textbf{Output}: All minimal hitting sets of $F$ up to the indicated size.

1. Let $H$ be the growing DAG and $L$ be the empty set. Generate a new node $n$, which is the
   root node of $H$, add it to $H$, let $label(n)$ and $h(n)$ be the empty set. Add $n$ to $L$, let $L'$ be the
   empty set and set $i = 0$.
2. For all nodes $n$ in $L$ do:
   (a) From left to right search for a set $C \in F$ such that $C \cap h(n)$ is the empty set. If there is
       no such set, a new minimal hitting set has been found and let $label(n) = \sqrt{.}$
   (b) Otherwise, for each $x \in C$ do:
      i. If there exists a previously handled node $m$ with $h(m) = h(n) \cup \{x\}$, then generate a
         new arc from $n$ to $m$.
      ii. Otherwise, generate a new node $n'$ with $h(n') = h(n) \cup \{x\}$, and an arc from $n$ to $n'$.
         If there exists a previously handled node $m$ with $label(m) = \sqrt{.}$, and $h(m) \subset h(n')$, 
         then close node $n'$ and let $label(n') = \times$. Otherwise, add $n'$ to $L'$.
   (c) Let $i = i + 1$.
3. If $L'$ is not empty, and $i \leq MAX$, let $L$ be $L'$ and $L' = \emptyset$, and go to 2.
4. Otherwise, return a set comprising $h(n)$ for all nodes $n$ with $label(n) = \sqrt{.}$

The hitting set algorithm obviously terminates for every finite set $F$ and $MAX$. If $F$ is not
empty, then the algorithm has at least two iterations. All hitting sets can be computed by setting
$MAX$ to the number of elements stored in $F$’s set, i.e., $MAX = |\bigcup_{x \in F} x|$.

\textbf{Example 8}. The hitting set DAG for $F = \{ \{12,14\}, \{12,13,15\}, \{12,14,16\}\}$ is given as follows
where the values of $h$ for each node are given under parentheses ($\{\}$):
The left-most node at the third level of the tree marked with $\times$ is closed because its corresponding hitting-set is not minimal. □

In the next section we present a diagnosis algorithm that combines the ATMS with the hitting set algorithm for computing all minimal diagnoses.

### 4.2 A diagnosis algorithm

From the definition of conflicts we know that they are a subset of the set of components leading to inconsistencies. From the ATMS we know that all environments leading to inconsistencies are stored in the label of the nogood. Hence, what we have to do is nothing else than introducing the $\neg AB$ predicates for components as assumptions and combining the nogood environments using the hitting set algorithm to obtain the diagnoses.

**ALGORITHM** MBD($SD,COMP,OBS$)

*Input:* the system model $SD$, the set of components $COMP$, and the set of observations $OBS$.

*Output:* a set of minimal diagnoses.

1. Generate a new ATMS
2. Add all rules and facts $r$ in $SD \cup OBS$ to the ATMS using PROPAGATE($r, \Phi, \{\} \})$.
3. Let $COSet$ be the label of the nogood of the ATMS.
4. Return AllMinHittingSets($COSet, \infty$).

The algorithm can be easily adapted to return only hitting sets of a certain size by replacing $\infty$ with a finite natural number. A 1 is for computing all single fault diagnoses, a 2 for all double fault diagnoses, and so on.

Note that the ATMS only accepts rules as inputs. Negated propositions or predicates like $\neg AB$ are not allowed. In order to solve this problem we have to replace all $\neg AB$ in $SD$ with $NAB$. Another issue is that the resulting hitting sets would be of the form $\{NAB(C_1), NAB(C_2), \ldots\}$. In order to get the diagnoses accordingly to the definition we have to select the names of the components only and to remove the surrounding $NAB$ predicates.

In the following section, we introduce an alternative definition of diagnosis where not only the correct component behavior but the faulty one is used to obtain diagnoses. Moreover, the approach allows for computing explanations from the models directly.
5 Abductive diagnosis

In this section we lay out the basic ideas of abductive reasoning and its application to diagnosis in a formal way. For this purpose, we first define a knowledge base comprising horn clause rules over propositional variables. The restriction of logical formula to be horn is not a real limitation in the context of physical systems since those models usually code the knowledge from causes to their effects. The used definitions are close the ones introduced by Friedrich at al. [FGN90] and others in the area of abductive diagnosis [CT90,CDT91].

In contrast to model-based diagnosis presented in a previous section, abduction makes use of models of faulty behavior when used for the purpose of diagnosis. However, abduction also makes use of the correct behavior but the reasoning is different. Instead using conflicts abduction searches for explanations of a given proposition. For example, consider the Nixon diamond (Example 1) again. If we are interested for an explanation that Nixon was not a pacifist, then we expect some root causes to be a good answer. In the case of the Nixon diamond, the answer would be that Nixon was member of the Republican party.

Let us now lay out the basic definitions of abductive reasoning. We start with the definition of a knowledge base.

**Definition 17 (Knowledge base (KB)).** A knowledge base (KB) is a tuple \((A, Hyp, Th)\) where

- \(A\) denotes the set of propositional variables,
- \(Hyp \subseteq A\) the set of hypothesis, and
- \(Th\) the set of horn clause sentences over \(A\).

In the definition of KB hypotheses corresponds directly to causes such that for every cause there is a propositional variable that allows to hypothesis about the truth value of the cause. Hence, we use the terms hypotheses and causes in an interchangeable way.

**Example 9.** The KB for the nixon diamond comprises the following parts:

\[
A = \{quaker(nixon), republican(nixon), pacifist(nixon), Default\_quaker(nixon), Default\_republican(nixon)\}
\]
\[
Hyp = \{Default\_quaker(nixon), Default\_republican(nixon)\}
\]
\[
Th = \{\begin{align*}
quaker(nixon) & \land Default\_quaker(nixon) \Rightarrow pacifist(nixon) \\
republican(nixon) & \land Default\_republican(nixon) \Rightarrow not\_pacifist(nixon) \\
pacifist(nixon) & \land not\_pacifist(nixon) \Rightarrow \bot
\end{align*}
\}
\]

For the purpose of diagnosis and explanation we need some observations. The KB together with the observations form the propositional horn clause abduction problem.

**Definition 18 (PHCAP).** Given a knowledge base \((A, Hyp, Th)\) and a set of observations \(Obs \subseteq A\) then the tuple \((A, Hyp, Th, Obs)\) forms a propositional horn clause abduction problem (PHCAP).

Given a PHCAP we define a solution like in [FGN90] as follows:
Definition 19 (Diagnosis; Solution of a PHCAP). Given a PHCAP \((A, \text{Hyp}, \text{Th}, \text{Obs})\). A set \(\Delta \subseteq \text{Hyp}\) is a solution (or diagnosis) if and only if \(\Delta \cup \text{Th} \models \text{Obs}\) and \(\Delta \cup \text{Th} \nvdash \bot\). A solution \(\Delta\) is parsimonious or minimal if and only if no set \(\Delta' \subset \Delta\) is a solution.

In this definition diagnoses need not to be minimal or parsimonious. In most practical cases only minimal diagnoses or minimal explanations for given effects are of interest. Hence, from here on we assume that all diagnoses are minimal diagnoses if not specified explicitly.

Example 10. Let us continue the Nixon diamond example adapted for abduction. In order to state the corresponding PHCAP we have to introduce observations to be explained. In our case we want an explanation for the fact that Nixon was not a pacifist.

\[
\text{Obs} = \{\text{not}_{-}\text{pacifist}(\text{nixon})\}
\]

Given the KB and \(\text{Obs}\) we are able to give a solution as explanation. From the rule

\[
\text{republican}(\text{nixon}) \land \text{Default}_{-}\text{republican}(\text{nixon}) \Rightarrow \text{not}_{-}\text{pacifist}(\text{nixon})
\]

we are able to infer \(\text{not}_{-}\text{pacifist}(\text{nixon})\) if the hypothesis \(\text{Default}_{-}\text{republican}(\text{nixon})\) is true. What we have to check is whether this assumption together with the theory leads to an inconsistency. This is not the case because only assuming that Nixon was both a \(\text{Default}_{-}\text{republican}\) and a \(\text{Default}_{-}\text{quaker}\) lead to an inconsistency. Therefore, \(\{\text{Default}_{-}\text{republican}(\text{nixon})\}\) is a solution and an explanation for the given observations. ∎

It is well known that the problem of finding minimal diagnoses for a given PHCAP is NP-complete. See for example Friedrich et al. [FGN90] for a proof. Moreover, in the same paper the authors describe a general diagnosis and therapy process that make use of a PHCAP directly in order to identify and correct the detected faulty behavior. Unfortunately the introduced algorithm cannot be used in our setting where exactly one diagnosis has to be identified before executing the necessary repair actions. Hence, instead of computing a diagnosis, checking its hypothesis, and if necessary modifying the corresponding PHCAP in order to obtain other diagnoses for further elaboration, in our case we have to compute all possible diagnoses and to reduce this set by adding new observations, which allow to discriminate diagnoses until one diagnosis is left and the corresponding treatment can be applied.

A reason for this change is that in the industrial context checking for correct hypothesis, i.e., mainly via replacing components, might be too expensive. Hence, the identification of one root cause allows for reducing the overall costs.

In order to compute discriminating additional observations we adopt the approach by De Kleer and Williams [dKW87] who introduced an algorithm for measurement selection, which is based on entropy. The original work of De Kleer and Williams is based on consistency-based diagnosis [Rei87] whereas our version is adapted for the abductive case.

The idea behind measurement selection is to identify an observations that help to restrict the search space as efficiently as possible. This is the case if an observation allows for dividing the set of diagnoses into two disjunctive subsets of almost the same size where one subset predicts the observations and the other does not. In this case search is done in a binary way, which requires only a logarithmic number of steps to find a unique solution.
Definition 20 (Discriminating observation). Given a PHCAP \((A, Hyp, Th, Obs)\) and two diagnoses \(\Delta_1\) and \(\Delta_2\). A new observation \(o \in A \setminus Obs\) discriminates two diagnoses if and only if \(\Delta_1\) is a diagnosis for \((A, Hyp, Th, Obs \cup \{o\})\) but \(\Delta_2\) is not.

The entropy \(H\) of an event is the product of the event’s probability and the logarithm of the event’s probability and is a measure of the information gain.

\[
H(o) = -p(o) \cdot \log_2 p(o) - (1 - p(o)) \cdot \log_2 (1 - p(o))
\]

The \(1 - p(o)\) part on the right of the equation is due to the fact that we also have to consider the observation to be \(\neg o\). The probability \(p(o)\) can be obtained using the number of diagnoses that predict the observation and the overall number of diagnoses.

\[
p(o) = \frac{|\{\Delta | \Delta \in \Delta\text{-Set}, \Delta \cup Th \models Obs \cup \{o\}\}|}{|\Delta\text{-Set}|}
\]

\(\Delta\text{-Set}\) is the set of diagnoses obtained from the original PHCAP, i.e., \(\Delta\text{-Set} = \{\Delta | \Delta \in Hyp, \Delta \cup Th \models Obs\}\).

Knowing \(H(o)\) for all possible \(o\) we are able to rank \(o\). The proposition \(o\) with the highest entropy \(H(o)\) is the one that should be looked at first. An abductive reasoning system would compute the entropy values and present them to the user. The user may select one of the proposition if it is true, i.e., can be observed and should be explained, and restart diagnosis again.

Although the use of entropy for probing was introduced by [dKW87] there are some differences. First, in [dKW87] the used diagnosis technique is consistency-based diagnosis where the correct behavior of a system is modeled. In abductive diagnosis the behavior in case of a fault has to be specified in order to extract the root cause. We refer the reader to [CDT91] where the relationship between abductive and consistency-based diagnosis is discussed in detail.

Second, in De Kleer and William’s approach [dKW87] all diagnoses and not only the minimal ones are used to compute the next best measurement location. In classical consistency-based diagnosis every superset of a diagnosis is itself a diagnosis and therefore minimal diagnoses characterize all possible solutions. This is not the case for abductive diagnosis where the minimal diagnoses do not characterize all possible diagnoses. Computing all possible diagnoses (and not only the minimal ones) and using them for probing seems therefore to be infeasible. A more detailed discussion about the use of fault models for diagnosis and its consequences can be found in [dKMR92].

Third, the introduced probing is not based on a specific type of model, e.g., a component-connection model. The entropy values for the propositions are directly computed. Since, there are several propositions, which are tightly coupled with a certain test, it makes sense to use such information within the diagnosis and probing cycle. The idea is not only to compute diagnoses for observations and stop but to use probing for gaining new observations and starting diagnoses again. This process ends when one single explanation can be derived.

5.1 Algorithm for abductive reasoning

In this section we discuss an algorithm for abductive reasoning that is based on the ATMS. We know that an ATMS ensures consistency, minimality, soundness, and completeness of the nodes’
labels. Nodes represent either propositions or assumptions. The latter are called hypothesis in the context of abduction. If we have now observations $\text{Obs} = \{o_1, \ldots, o_k\}$ we can easily compute all possible explanations. For this purpose, we introduce a new rule of the form

$$o_1 \land \ldots \land o_k \Rightarrow \text{explain}$$

where $\text{explain}$ is a proposition that is not used in any other rule, and send it to the ATMS using the \text{PROPAGATE} function. The ATMS computes a label for $\text{explain}$ comprising environments. For every environment $E$ of the label of $\text{explain}$ we know that $E \cup Th \models \text{explain}$ and $E \cup Th \not\models \bot$ must hold because of the ensured label properties. Therefore, every environment of $\text{explain}$ is an abductive diagnosis (or solution) of the PHCAP. The following algorithm summarizes the findings to implement abductive diagnosis.

\begin{algorithm}
\textbf{ALGORITHM ABDUCTIVE\_DIAGNOSIS}\((Th, Hyp, Obs)\)
\begin{algorithmic}
\STATE \textbf{Input:} the system model $Th$ where $\text{explain}$ is not used as proposition and hypothesis, the set of hypothesis $Hyp$, and the set of observations $Obs$.
\STATE \textbf{Output:} a set of minimal diagnoses.
\STATE 1. Generate a new ATMS
\STATE 2. Add all rules and facts $r$ in $Th$ to the ATMS using $\text{PROPAGATE}(r, \Phi, \{\}\}$. 
\STATE 3. Assume that $Obs = \{o_1, \ldots, o_k\}$. Add the rule $o_1 \land \ldots \land o_k \Rightarrow \text{explain}$ to the ATMS using $\text{PROPAGATE}$. 
\STATE 4. Return the label of $\text{explain}$ as result.
\end{algorithmic}
\end{algorithm}

We illustrate the use of the \textbf{ABDUCTIVE\_DIAGNOSIS} algorithm using the inverter chain example (Example 2).

\textbf{Example 11.} For a model of the inverter chain example from Example 2 we modify the underlying model slightly. We introduce fault models stuck-at-0 and stuck-at-1 indicating that an inverter only delivers 0 or 1 respectively in case of a fault ignoring the input. The resulting rules of this modification are:

\begin{align*}
\forall X : \text{inverter}(X) &\rightarrow (\text{NAB}(X) \land \text{in}(X, 0) \rightarrow \text{out}(X, 1)) \\
\forall X : \text{inverter}(X) &\rightarrow (\text{NAB}(X) \land \text{in}(X, 1) \rightarrow \text{out}(X, 0)) \\
\forall X : \text{inverter}(X) &\rightarrow (\text{NAB}(X) \land \text{out}(X, 0) \rightarrow \text{in}(X, 1)) \\
\forall X : \text{inverter}(X) &\rightarrow (\text{NAB}(X) \land \text{out}(X, 1) \rightarrow \text{in}(X, 0))
\end{align*}

\begin{align*}
\forall X : \text{inverter}(X) &\rightarrow (S0(X) \rightarrow \text{out}(X, 0)) \\
\forall X : \text{inverter}(X) &\rightarrow (S1(X) \rightarrow \text{out}(X, 1))
\end{align*}

The other rules for stating the structure remains the same. Note that before sending these rules to the ATMS we have to ground them by replacing all variables $X$ with the names of the inverters, i.e., $I_1, \ldots, I_4$. Moreover, the rules have to be changed to form implications, i.e., we
have to separate rules of the form \( x \leftrightarrow y \) to \( x \rightarrow y \) and \( y \rightarrow x \). The predicates \( NAB \), \( S0 \), and \( S1 \) are hypothesis of the abduction problem representing the root causes correctness, stuck-at-0, and stuck-at-1 for each inverter.

In order to explain a certain observation at the output, we also have to state the stimulus, i.e., the input as logical rule. For example, we add \( \rightarrow val(a, 1) \) to the theory and to the ATMS when we want to solve the problem related to the stimulus \( val(a, 1) \). The ATMS can now be used to explain the observation \( Obs = \{ val(e, 0) \} \). One possible explanation is \( \{ S0(I4) \} \) and another one \( \{ S0(I1), NAB(I2), NAB(I3), NAB(I4) \} \).

6 Conclusions

In this article we presented the ATMS as general reasoning system that can be used for implementing diagnosis based on the correct behavior of systems and abductive reasoning. The latter is also used for explanation purposes and is able to handle fault models. The given algorithms are not optimal from the computational point of view and in practice improvements are necessary especially when dealing with larger systems. The basic idea behind the article is to introduce the reader into logic based methods for diagnosis, predication, and explanation under a general framework.

In summary, the article provides the following contributions:

1. The ATMS is a system that allows to retain consistency and implements a restricted form of default reasoning usually used in common sense reasoning. The label update used in the ATMS is NP-complete.
2. Model-based diagnosis uses a model of the system and given observations directly to identify root causes of a detected misbehavior. The root causes are called diagnoses. The used models specify the correct behavior of components and the structure of the system. The ATMS can be used for computing the diagnoses. For this purpose the label of the nogood has to be transformed. This is done by computing the hitting set of all environments in the label. The hitting sets correspond to the diagnoses.
3. Abductive reasoning also makes use of a model. The model should capture the cause-effect relationships of a system. Beside the correct behavior fault models can also be used. In abductive reasoning the input of a system, i.e., the stimulus, has to be stated as facts. Observations, e.g., the outputs of a system or intermediate values, are explained using the given model. Again the ATMS can be used for computing all abductive explanations. We only need to introduce a new rule stating that the observations lead to a proposition \( explain \) and use the ATMS to compute a label for \( explain \). This label comprises all possible abductive explanations.

References


